The property interception code snippets in previous topics were all examples of what are termed **Named interceptor actions**, in that they each specified a single specific named property to be intercepted. It is also possible to create **Unnamed interceptor actions** that apply to all of the properties for a specific target type. For example, suppose that the following code were implemented in the Employee partial class:

```
[BeforeSet]
public void BeforeSetAny(IbCore.IPropertyInterceptorArgs args) {
 if (!Thread.CurrentPrincipal.IsInRole("Administrator")) {
   throw new InvalidOperationException("Only admistrators can change Product data!");
  }
}
```

```
<BeforeSet> _
Public Sub BeforeSetAny(ByVal args As IPropertyInterceptorArgs)
 If Not Thread.CurrentPrincipal.IsInRole("Administrator") Then
   Throw New InvalidOperationException( _
     "Only admistrators can change data")
 End If
End Sub
```

The result of this code would be that only those users logged in as administrators would be allowed to call any property setters within the Employee class.

A similar 'after set' action might look like the following:

```
[AfterSet]
public void AfterSetAny(IbCore.IPropertyInterceptorArgs args) {
  LogChangeToCustomer(args.Instance);
}
```

```
<AfterSet> _
Public Sub AfterSetAny(ByVal args As IPropertyInterceptorArgs)
  LogChangeToEmployee(args.Instance)
End Sub
```

This would log any changes to the employee class.

Later in this document we will also describe how to define interceptors that apply across multiple types as well as multiple properties within a single type.